UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Redundant Firewalls for Web Applications

Dauto Ussene Jeichande

Mestrado em Segurança Informática

Dissertação orientada por:
Prof. Doutor Hugo Alexandre Tavares Miranda

2016

Abstract

The websites of small, medium and large companies are a great competitive advantage in the business world. Companies invest a considerable amount of money to ensure they appear on top of the ranks when users seek for goods and services in search engines. On the other hand, websites open yet another door for an adversary to compromise the security of the organizations.

The conventional network firewalls and intrusion prevention systems have limitations in protecting web applications because they operate at the network layer. The Web Application Firewall (WAF) is a solution that companies use to mitigate these limitations. The difference is in the ability to analyze the logic of the application layer. The WAF not only detects known attacks on the web application environment but can also detect and prevent new types of attacks through pattern analysis to traffic.

This project aims to implement a redundant WAF architecture using the concept of diversity at the operating system (OS), web server and WAF technology. To evaluate the solution proposed we used vulnerabilities scanner tools to execute a set of security tests and a web load tests tool to measure the response times.

The evaluation compared the resilience and impact on the performance of our architecture with several other configurations, with and without WAFs.

Keywords: Web Application Security, Web Application Firewall, Reverse Proxy, Diversity, Load Balancing

Resumo

Nos dias de hoje, as páginas de Internet das pequenas, médias e grandes empresas são uma grande vantagem competitiva no Mundo de negócios. As empresas investem consideravelmente para garantir que aparecem em primeiro lugar quando os utilizadores pesquisam por bens e serviços nos motares de busca.

Por outro lado, os adversários usam este tipo de acesso para tentar comprometer a segurança das organizações. Os filtros de rede convencionais e detetores de intrusões apresentam limitações na proteção das aplicações web porque eles inspecionam ao nível da camada de rede.

Os filtros para aplicações web (WAF) são uma solução que as empresas usam para mitigar estas limitações. A diferença está na habilidade de analisar a lógica da camada aplicacional. As WAFs não detetam somente ataques conhecidos no ambiente aplicacional web como também e quando devidamente configuradas podem detetar e prevenir novos tipos de ataques através da análise de padrões do tráfego.

Neste projeto é implementada uma arquitetura de WAFs redundantes utilizando o conceito de diversidade a nível do sistema operativo, servidor web e da tecnologia WAF. Para avaliar o desempenho usamos ferramentas de análise de vulnerabilidades para executar testes de segurança e uma ferramenta de testes de carga para aplicações web para medir os tempos de resposta.

A avaliação comparou resiliência da nossa arquitetura e o impacto no desempenho com outras configurações, com e sem WAFs.

Palavras-chave: Segurança de Aplicações Web, Firewalls Aplicacionais Web, Servidor Intermediário, Diversidade, Balanceamento de Tráfego

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The use of web business applications is increasing in organizations. They are very easy to access through the ubiquitous web browser available on any computer and a great competitive advantage in the business world. This web access is also available for adversaries, which use it as an attack surface. According to a survey performed by Symantec, from 2012 to 2013, the proportion of scanned websites with vulnerabilities increased from 53 % to 77 % [24].

The conventional network firewalls have limitations protecting web applications because they were designed with two different approaches: stateless or stateful packet filtering. Stateless packet filtering firewalls analyze the network and transport headers of the inbound and outbound packets. In contrast, stateful packet firewalls track the packets connections by maintaining a state table. This table is used to determine if traffic is part of an established connection.

The Intrusion Prevention Systems (IPS) also have restrictions when used to protect specific web applications. IPS solutions operate at the network layer and can detect attacks such as stealth port scans and attacks aimed at protocols like Secure Socket Shell (SSH). They allow or deny packets after comparing them to known attack signatures. IPS has no knowledge of the web application layer constructs, the data structure, and encoding. For this reason, IPSs fail to prevent many attacks, or generates false positives, depending on the security policies.

The WAF is a solution to solve all of the above. It can completely analyze the logic of application layer and apply strict security checks on the decoded request content. It can detect known attacks and even new types of attacks by analyzing the pattern of the traffic [10].

This project aims to evaluate the contribution of redundant WAFs with a diversity approach regarding the OS, web server and the WAF technology. Redundancy is experimented as a tool to provide protection with both diversity and high availability capabilities for the web business applications. It is often argued that redundancy without diversity is useless against organized and systematic attack as most of the adversaries take

in consideration the predictable behavior of the target applications. If the redundant nodes have the same characteristics, the same exploit might work on the entire architecture. With our solution, if the adversary succeeds to break one node, there is a low probability of compromising the remaining redundant nodes [3]. The concept of security through diversity is a topic of interest these days in the design of intrusion tolerance architectures.

1.1 Motivation

One of the organization's operational goals is to maximize the uptime of web business applications. To achieve this objective one of the operation tasks is to protect these applications against the increased web threats.

The presented work proposes a WAF architecture based on apache and nginx web servers which are the most popular web servers [23]. It is expected that most of the companies have a well-established experience with apache and nginx and can benefit from a smooth integration of this solution on their environments.

This work will contribute to evaluate the implementation of redundant WAFs with a design diversity. Design diversity is the typical form of parallel redundancy for fault tolerance against design faults (either accidental or intentional): the multiple replicas of the system are handled by diverse software components [3]. This project uses two different WAFs, but the same strategy can be applied if the number of nodes increases.

1.2 Objectives

The project has the following objectives:

- Define a redundant WAF architecture composed by nodes with diverse OS, web server and WAF technology to reduce the probability of having common vulnerabilities
    - Configure the load balancing. When one of the servers is down for maintenance or unpredictable reason the web applications should still be available for the end users
    - Define rules to protect the development track solution from web attacks
    - Generate a whitelist to avoid false positives
- Test and evaluate the solution designed
    - Compare the resilience of our architecture with other configurations, with and without WAFs.
    - Compare the impact on the performance of our architecture with other configurations, with and without WAFs.

1.3 Document Structure

The rest of this dissertation is organized as follow.

Chapter 2 – Related Work. Reviews the problems identified by other authors and the solutions proposed by them. Then it presents the new goals this project wants to accomplish. Provides an overview of the top 10 web application security risks and describes the tools and concepts used in this work.

Chapter 3 – Architecture Implementation. This chapter describes the installation and configuration of all architecture components such as servers, firewalls, load balancers, and software.

Chapter 4 – Tests and Performance Evaluation. This part of the document presents the results and evaluation of the web security and web load stress tests executed in the testbed.

Chapter 5 – Conclusions and Future Work. Summarizes what was done with this work. Describes open issues and directions for the continuation of this work.

Chapter 2

Related work

There are some works related to web server's architectures which used the concept of redundancy, diversity, and rejuvenation in order to increase the system resilience to attacks [18] [19] [4].

A generic architecture where redundant proxies filter client requests to a redundant group of diversified application servers was proposed in [18]. The authors implemented adaptive redundancy to improve performance. With this method of operation, the level of redundancy depends on the current alert level. When the attack density increases, the number of web servers that process each client request will also increase and consequently the performance will degrade. The case study was based on a travel agency web infrastructure.

In [19], authors focused on the study of different intrusion-tolerant architectures for web servers (based on intelligent adaptive reconfiguration). The objective is to help to build a more secure and resilient server system.

In [4], the authors argue that the use of diversity and redundancy can make authentication systems resistant to unknown vulnerabilities. The redundancy is employed in a way that even if some are successfully attacked, the authentication system is not compromised. The diversity is introduced in authentication mechanisms so that they can resist attacks that exploit new vulnerabilities.

Similar to our work, Both authors in [18] and [19] propose solutions to protect a web application. The detection of those architectures is based on Snort Intrusion Detection System (IDS) installed on proxies which have limitations in analyzing the application layer. In contrast, this project focuses on providing availability and better performance in the prevention of web attacks by implementing redundancy and diversity in WAF proxies.

2.1 Diversity

The diversity approach in this architecture was applied on OS, web server, and WAF technology.

2.1.1 Operating system

The table below shows the vulnerabilities correlation between two different OS extracted from the study reported in [7]. CentOS, an equally popular Linux Distribution was not included in the study. However, it should be noted that CentOS built from much of the Red Hat Enterprise Linux codebase. The table shows that Ubuntu and Red Hat have one common vulnerability in system software making them good candidates for comparison in a diversity experiment. To build the table below the author's correlated vulnerabilities that were shared by OS pairs in the period between 1994 and 2011. These vulnerabilities were collected from Nationality Vulnerability Database.

Operating system pairs	Driver	Kernel	Sys. Soft.	Total
Win2000-Win2003	0	42	43	85
Win2003-Win2008	0	18	22	40
OpenBSD-FreeBSD	1	14	18	33
NetBSD-FreeBSD	2	13	10	25
OpenBSD-NetBSD	1	8	8	17
Win2000-Win2008	0	8	8	16
Debian-Red Hat	0	5	8	13
NetBSD-Solaris	0	4	6	10
FreeBSD-Solaris	0	5	3	8
OpenSolaris-Solaris	0	3	3	6
OpenBSD-Solaris	0	5	1	6
Solaris-Red Hat	0	3	3	6
NetBSD-Red Hat	0	0	6	6
FreeBSD-Red Hat	0	1	4	5
OpenBSD-Red Hat	0	1	3	4
NetBSD-Debian	0	0	4	4
FreeBSD-Win2000	1	3	0	4
OpenBSD-Win2000	0	3	0	3
NetBSD-Win2000	1	2	0	3
Solaris-Win2000	0	3	0	3
OpenBSD-Win2003	0	2	0	2
FreeBSD-Win2003	0	2	0	2
Solaris-Debian	0	1	1	2
Debian-Ubuntu	0	0	2	2
NetBSD-Win2003	0	1	0	1
NetBSD-Win2008	0	1	0	1
FreeBSD-Debian	0	0	1	1
FreeBSD-Win2008	0	1	0	1
Debian-Win2000	0	0	1	1
Ubuntu-Red Hat	0	0	1	1
Ubuntu-Win2000	0	0	1	1
Red Hat-Win2000	0	0	1	1
OpenBSD-Win2008	0	1	0	1
Solaris-Win2003	0	1	0	1
Debian-Win2003	0	0	1	1
Ubuntu-Win2003	0	0	1	1
Red Hat-Win2003	0	0	1	1

Table 1: Common Vulnerabilities on Isolated Thin Servers [7]

Considering the minimal number of common vulnerabilities and the fact that they are both Linux distribution this work opted for using Ubuntu and CentOS.

2.1.2 Web server

The web servers addressed in our work are the most popular namely apache and nginx [23]. As of 17/11/2015, Nginx 1.9.5 has no reported vulnerabilities [23] which mean that there is a low probability of sharing the vulnerabilities reported in apache 2.4.7 [23].

2.1.3 Web Application Firewall

A WAF is an appliance, server plugin, or filter that applies a set of rules to a Hyper-Text Transfer Protocol (HTTP) conversation. The rules protect the web application from the most common attacks such as injection, broken authentication and session management, Cross-Site Scripting (XSS), insecure direct object references, Cross-Site Request Forgery (CSRF) and invalidated redirects and forwards.

Most of the open source WAF products are based on modsecurity and naxsi. The other reason of using these WAF solutions is the fact that modsecurity and naxsi integrate very well with apache and nginx respectively. Naxsi 0.54 and modsecurity 2.7.7 have no reported vulnerability on 17/11/2015 [23]. Consequently, an adversary will have more difficulties in compromising the web applications by exploiting a WAF vulnerability. It is expected that using other WAF solutions and combine them to have more diversified nodes will improve the web security. The number of diversified nodes will depend on the security requirements.

Modsecurity

Modsecurity is an open source and free WAF designed to integrate with apache and nginx. Functionalities offered by modsecurity falls roughly into four areas:

- Parsing: the supported data formats (E.g.: XML) are backed by parsers that extract bits of data and record them for the use of the rules
- Buffering: both request and response bodies are buffered. This allows modsecurity to sees complete requests before they are forwarded to the application for processing, and complete responses before they are delivered to clients
- Logging: the log of transaction data can be customized. This allows to record complete HTTP traffic and which transaction are recorded.
- Rule Engine: the HTTP traffic stream is inspected in real-time according to defined rules. It allows mitigating application vulnerabilities in a separate layer by applying block rules while developing a patch to fix the application code. This feature is known as virtual patching

Modsecurity operates in self-contained or anomaly scoring detection mode. In a self-contained mode where if a rule triggers, it will execute any disruptive/logging actions

specified in the current rule. This approach offers better performance (lower latency/resources) because the first disruptive match will stop further processing. However, lower severity alerts are largely ignored. Alternatively, modsecurity could be configured in anomaly scoring detection mode which each matched rule will not block, but rather will increment anomaly scores using modsecurity's setvar action. If the current transactional score is above a defined threshold, the request will be denied.

2.1.3.1 Naxsi

Naxsi is an open-source and free WAF for nginx which provides security through a whitelist system, also known as “positive model firewalling”. Below are presented the most significant features:

- Parsing: parses types of requests (E.g.: get, put and post) and body content-types (E.g.: application/x-www)
- Logging: records transaction using a standard format.
- Rule Engine: inspects the HTTP traffic stream in real-time according to defined rules. The action to accomplish can be to block (production mode) or to record (learning mode). The learning mode helps the generation of whitelists to prevent blocks from occurring in production mode. This is one of the reasons to use naxsi as another WAF node

Naxsi uses a file configuration with scores assigned to each rule. When the rule matches and a score exceeds a predicted threshold, the request is blocked. Figure 1 presents the default scores configuration.

```
LearningMode;
#SecRulesEnabled;
DeniedUrl "/RequestDenied";
## Check & Blocking Rules
CheckRule "$SQL >= 8" BLOCK;
CheckRule "$RFI >= 8" BLOCK;
CheckRule "$TRAVERSAL >= 4" BLOCK;
CheckRule "$EVADE >= 4" BLOCK;
CheckRule "$XSS >= 8" BLOCK;
CheckRule "$UPLOAD >= 8" BLOCK;
```

Figure 1: Naxsi Score Rules

The table below compares the WAFs taking in consideration the naxsi and modsecurity features.

Features	Naxsi	Modsecurity
Parsing	☑	☑
Buffering	☐	☑
Logging	☑	☑
Rule Engine	☑	☑
Virtual Patching	☐	☑
Learning mode	☑	☐

Table 2: Naxsi and Modsecurity Comparison

The notable main differences are in learning mode, buffering and virtual patching features.

2.2 Web Attacks Classification

The web attacks target is the application and focuses on the layer 7 of the Open Systems Interconnection (OSI) model. This section describes the most common types of attacks relevant to web applications [16].

2.2.1 Injection

Injection flaws, such as SQL and OS injection occur when untrusted and malicious data is sent to an interpreter as part of a query or command. The objective is to trick the interpreter into executing unintended commands or accessing data without proper authorization.

2.2.2 Broken Authentication and Session Management

The complexity of designing an authentication and session management scheme that adequately protects credentials in all aspects of the web applications sometimes is an underestimate. Authentications and session management mechanisms may allow collecting passwords, keys or session tokens to assume user's identity.

2.2.3 Cross-Site Scripting (XSS)

In XSS attacks, the application accepts untrusted data and sends it to a web browser without proper validation or escaping. XSS allows the adversaries to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

2.2.4 Insecure Direct Object References

A direct object reference occurs when application exposes a reference to an internal implementation object, such as a file, directory, or database key. The adversary exploits this reference to bypass authorization and access resources in the system directly.

2.2.5 Cross-Site Request Forgery (CSRF)

A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests which the vulnerable application thinks are legitimate requests from the victim.

2.2.6 Unvalidated Redirects and Forwards

Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

2.3 Discussion

Web-based solutions are being implemented for nearly every aspect of business operations. Web servers are also increasingly the primary source for malware delivery networks, hosting malware and putting users, resources and reputations at risk.

According to web application vulnerability statistics from Positive Technologies [34] the most common vulnerability of 2013 namely Cross-Site Scripting (XSS), was second most common in 2014, affecting 70% of web applications analyzed. The ten most common vulnerabilities also include SQL Injection, a critical vulnerability detected in 48% of the web resources studied as showed in the figure.

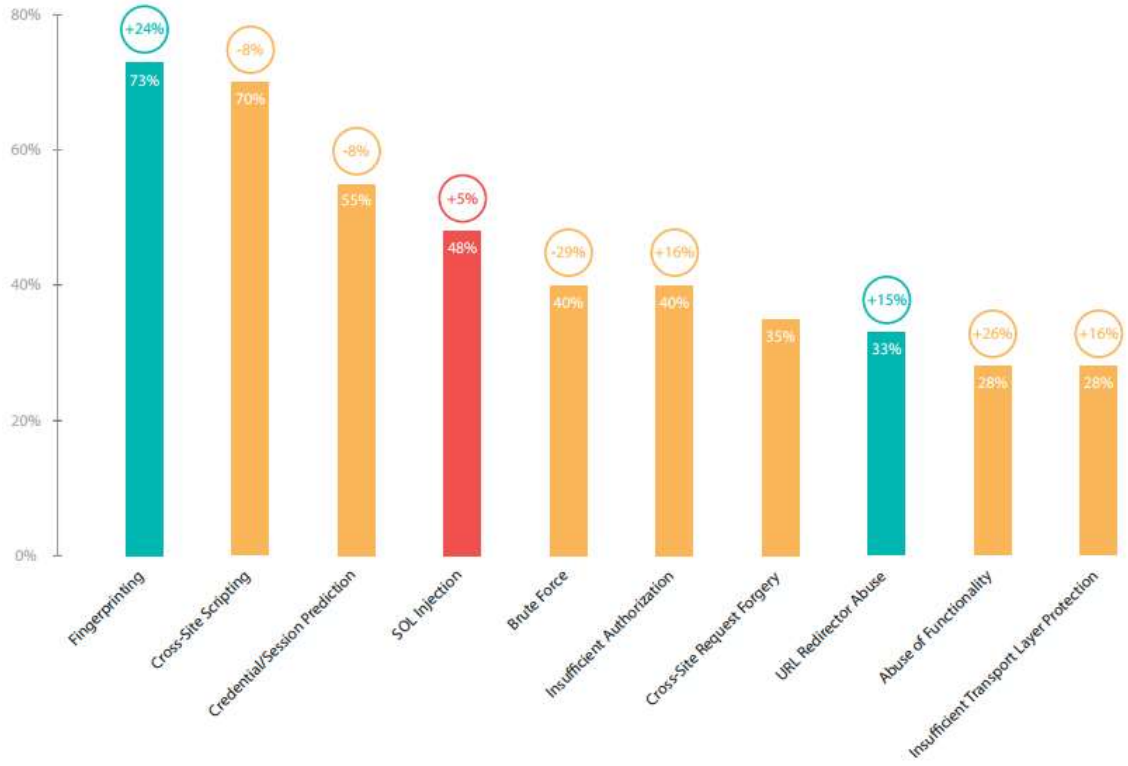

Figure 2: Web Application Vulnerability Statistics 2014

To protect the web application against these attacks some commercial and open source WAFs are available. This work focuses on open source solutions namely modsecurity and naxsi.

It is common to use redundancy when designing the architecture with the objective to avoid a single point of failure. From a security perspective diversity, enables a more robust protection because the adversary might need to combine multiple techniques to compromise the target. As of 29th June 2016, there is no Commercial Off-The-Shelf

(COTS) product that implements diversity concept. This work contributes to fulfill this gap.

Chapter 3

Architecture Implementation

3.1 Architecture

The implementation consisted of installation and configuration of all architecture components such as servers, firewalls, load balancers, and software. This project was developed using the Antex international web infrastructure[1]. The original web architecture depicted in figure 1:

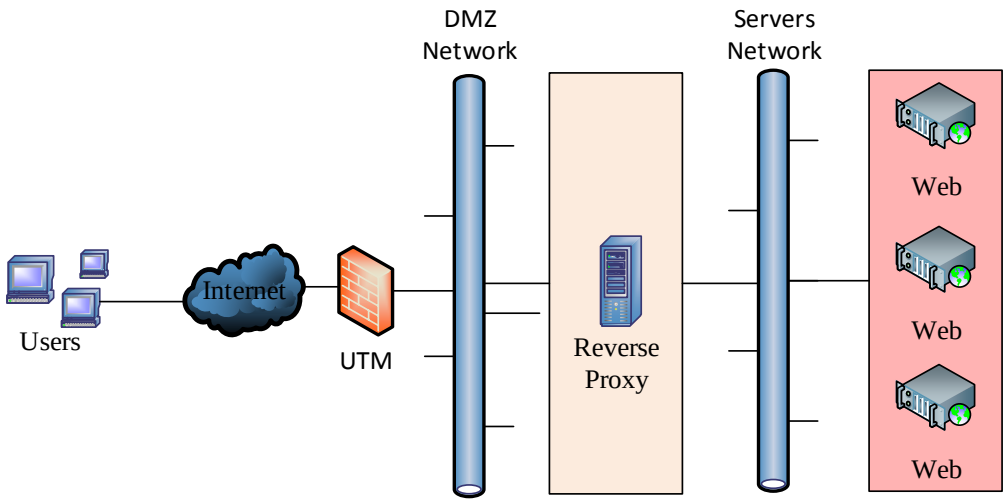

Figure 3: Antex International Current Web Architecture

[1] https://moldtrack.ai-emea.com

In this architecture, a reverse proxy is located behind a Unified Threat Management (UTM) and in front of web servers. A UTM include functions such as network firewalling and intrusion detection and prevention. All incoming requests are delivered to the reverse proxy which then forwards them to the back end web servers on behalf of the originating client application. In the current architecture, the traffic is inspected for malicious content by the IPS.

The new architecture was designed taking into consideration the original web infrastructure of the Antex International organization. Figure 3 illustrates the architecture designed in the scope of this work.

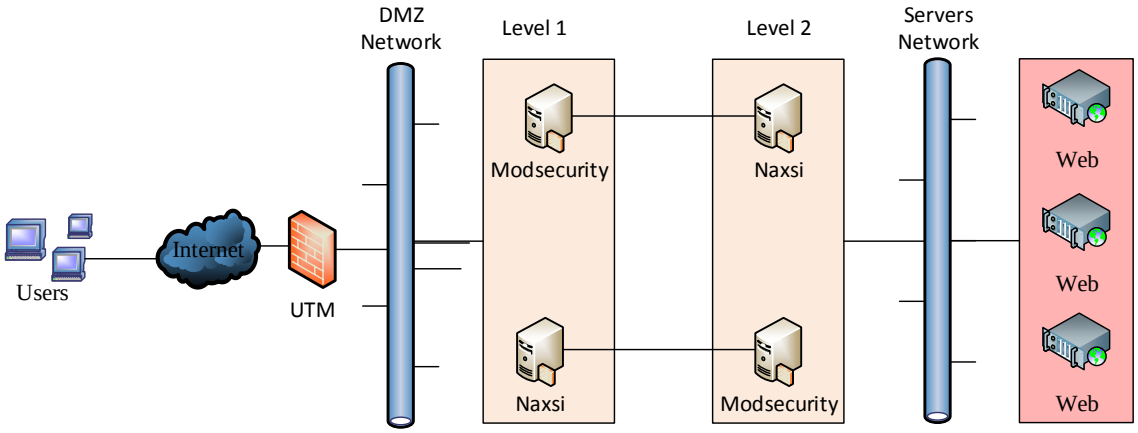

Figure 4: Antex International Proposed Web Architecture

All the components are deployed between a Unified Threat Management (UTM) and the web servers. All HTTP traffic is flowing through the WAFs. Incoming requests are addressed to the WAFs, who are responsible for forwarding them to the back-end web servers on behalf of the originating client. The UTM equally plays the role of load-balancer, distributing the load by the two "Level 1" WAFs. Each of the "Level 1" WAFs is connected to a single "Level 2" WAF of the opposing model, thus enforcing all the requests to be evaluated by the two WAF models. The "Level 2" WAFs send each of the web requests to be processed by the web application.

It should be noted that WAFs use the reverse proxy mode, what permits to terminate connections if an attack is detected. As will be discussed in the tests and performance evaluation section, this option has the drawback of notably increasing latency.

3.2 Configuration

This section describes the configuration performed on the architecture components.

3.2.1 Hardware

The servers, firewalls and load balancer are virtualized. For the test environment, the virtual machines were created on the same physical host. For production mode, we recommend installing the virtual machines in different physical hosts to avoid the creation of a single point of failure. VirtualBox was selected as hypervisor because it is the only virtualization solution available for Linux and Windows, simplifying the migration between these operating systems.

3.2.2 Server

Each WAF is run on a virtual server emulating an Intel Xeon 2.4GHz with 2 GB of RAM and a 30 GB hard drive.

Table 3 lists the software installed on the servers which was the last stable release of each package as of 30/11/2016.

Server	**OS**	**Web server**	**WAF**
Naxsi01	Centos 7.1.1503	Nginx 1.9.5	Naxsi 0.54
Naxsi02	Centos 7.1.1503	Nginx 1.9.5	Naxsi 0.54
Modsecurity01	Ubuntu 14.04.3 LTS	Apache 2.4.7	ModSecurity 2.7.7
Modsecurity02	Ubuntu 14.04.3 LTS	Apache 2.4.7	ModSecurity 2.7.7

Table 3: Software

The operating system was installed with the LVM (Logical Volume Manager) option enabled to facilitate the partitions dimensioning. The var directories were created in the independent partition to avoid OS failures resulting from hard drive space exhaustion. This is expected as WAFs consume a considerable amount of disk space writing detection logs.

The business application used for this test bed was Redmine. Redmine is an open source project management web application, written using the Ruby on Rails framework. We created a clone of the current production virtual machine used for software development tasks.

3.2.3 Nginx - Naxsi

Naxsi was added as a module when precompiling nginx during the installation process. The command nginx –V shows the version and configuration options:

```
$ nginx -V
nginx version: nginx/1.9.5
built by gcc 4.8.3 20140911 (Red Hat 4.8.3-9) (GCC)
built with OpenSSL 1.0.1e-fips 11 Feb 2013
TLS SNI support enabled
configure arguments: --user=root --group=root --
prefix=/usr/share/nginx --sbin-path=/usr/sbin/nginx --conf-
path=/etc/nginx/nginx.conf --error-log-path=/var/log/nginx/error.log
--http-log-path=/var/log/nginx/access.log --http-client-body-temp-
path=/var/lib/nginx/tmp/client_body --http-proxy-temp-
path=/var/lib/nginx/tmp/proxy --http-fastcgi-temp-
path=/var/lib/nginx/tmp/fastcgi --pid-path=/var/run/nginx.pid --
lock-path=/var/lock/subsys/nginx --with-http_ssl_module --with-
http_realip_module --with-http_addition_module --with-
http_sub_module --with-http_dav_module --with-http_flv_module --
with-http_gzip_static_module --with-http_stub_status_module --with-
http_perl_module --with-ld-opt=-Wl,-E --with-http_geoip_module --
add-module=../naxsi-master/naxsi_src
```

Figure 5: Nginx Installation Information

Naxsi configuration includes the nasxi core rules [9] and complemented with doxi rules [25]. The nasxi core rules [9] included in this main configuration file are the default rules to prevent SQL injections, remote file inclusion, directory traversal, cross-site scripting, evasion and file uploads attacks. The doxi rules [24] are available as an independent Github repository and aim to provide additional protection against this kind of attacks.

We added to naxsi score configuration the following entries for doxi rules: `CheckRule "$UWA >= 8" BLOCK` and `CheckRule "$ATTACK >= 8" BLOCK`. These scores are included in the default configuration.

Naxsi was initially configured in “LearningMode”. With this setup, the malicious requests are registered in the error log file and not blocked. An example of the whitelist produced by learning mode is presented in Appendix A.

Figure 5 shows an example of a false positive. This request is detected as RFI attack because it includes the string https in the middle of the URL and the score matches the value 8 specified in the RFI /etc/nginx/naxsi.rules file.

```
2016/02/18 12:41:22 [error] 2988#0: *1 NAXSI_FMT:
ip=46.20.43.122&server=moldtrack.ai-
emea.com&uri=/login&learning=0&vers=0.54&total_processed=1&total_blo
cked=1&block=1&cscore0=$RFI&score0=8&zone0=ARGS&id0=1101&var_name0=b
ack_url, client: 46.20.43.122, server: moldtrack.ai-emea.com,
request: "GET /login?back_url=https%3A%2F%2Fmoldtrack.ai-emea.com%2F
HTTP/1.1", host: "moldtrack.ai-emea.com"
```

Figure 6: False Positive - RFI

During the creation of the whitelist, we had to make sure that only legitimate traffic is presented to the nginx. This was accomplished by using the firewall to restrict access to the application from the IP address producing the legitimate learning traffic.

The nxapi tool which is designed to import the naxsi events from this whitelist logfile into an elastic search database:

If the data is added correctly, the nxtool will be able to generate statistics regarding the client as well as the URLs accessed. An example of these statistics is depicted in figure 6:

```
$./nxtool.py -c nxapi.json -x
# size :1000
# Whitelist(ing) ratio :
# false 49.78 % (total:452/908)
# Top servers :
# moldtrack.ai-emea.com 100.0 % (total:452/452)
# Top URI(s) :
# /projects 14.16 % (total:64/452)
# /projects/show/antex-infra 14.16 % (total:64/452)
# /issues/show/1219 7.08 % (total:32/452)
# /projects/antex-infra/issues 7.08 % (total:32/452)
# /projects/show/hrpm-invoicet 7.08 % (total:32/452)
# /issues/edit/1219 4.42 % (total:20/452)
# / 3.54 % (total:16/452)
# /images/ticket_note.png 3.54 % (total:16/452)
# /my/account 3.54 % (total:16/452)
# /my/page 3.54 % (total:16/452)
# /projects/activity/antex-infra 3.54 % (total:16/452)
# Top Zone(s) :
# HEADERS 99.12 % (total:448/452)
# BODY 0.88 % (total:4/452)
# Top Peer(s) :
# 46.20.43.122 100.0 % (total:452/452)
```

Figure 7: Nxapi – Display Database Content

The configuration including the default, doxi and whitelist rules was tested using penetration-testing tools. The results of these experiments are described in the tests and performance evaluation section.

3.2.4 Apache – Modsecurity

Modsecurity was installed from Ubuntu repository.

Modsecurity was enabled as an apache2 module and the OWASP ModSecurity Core Rule Sets (CRS) [13] were included in apache main configuration file.

The OWASP ModSecurity CRS provides protections for the following attack/threat categories:

- HTTP protection: detecting violations of the HTTP protocol and a locally defined usage policy
- Real-time blacklist lookups: which utilizes 3rd party IP reputation
- HTTP denial of service protections: defense against HTTP flooding and slow HTTP DoS attacks
- Common web attacks protection: detecting common web application security attacks such as SQL injections, XSS and CSRF

- Automation detection: detecting bots, crawlers, scanners and another surface malicious activity
- Integration with antivirus scanning for file uploads: detects malicious files uploaded through the web application
- Tracking sensitive data: tracks credit card usage and blocks leakages
- Trojan protection: detecting access to trojans horses
- Identification of application defects: alerts on application misconfigurations
- Error detection and hiding: disguising error messages sent by the server

Modsecurity was configured in a self-contained mode. The self-contained mode is in contrast with naxsi mode as the latter follows a scheme where each matched rule adds to the request score and requests are dropped if their score is above a predefined threshold. In addition, naxsi redirects dropped requests to "RequestDenied" page while modsecurity returns a page with "Forbidden" message (code 403).

Finally, modsecurity was configured to allow some content types normally to be uploaded by the users to redmine and to ignore some rules that aim to prevent attacks out of the scope of this project namely cold fusion and LDAP injection.

3.2.5 Load balancer

The load balancer service is integrated into the UTM appliance. We used one load balancer to distribute HTTP traffic between the level 1 WAFs (naxsi and modsecurity).

A monitor for HTTP, which constantly send requests to the nodes to check the availability was defined. A node is considered alive when the response code is 200 OK.

The monitor configuration was applied in two pools, one for HTTP and the second for HTTPS service to check the nodes availability. Each pool is composed of the level l WAFs with IP address 10.0.1.27 and 10.0.1.50.

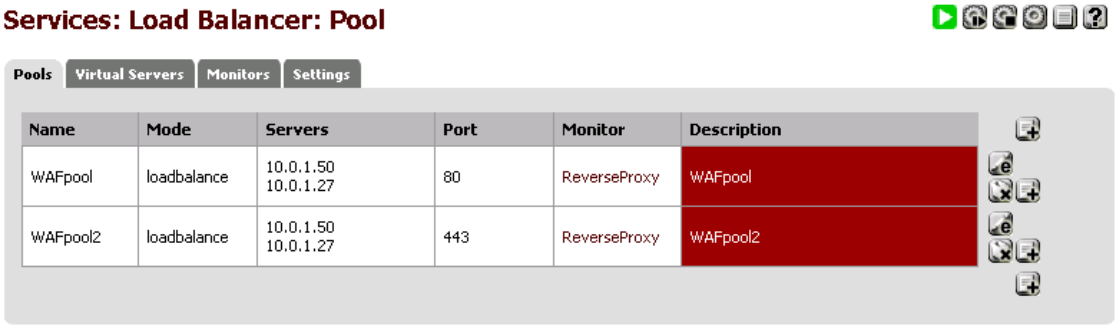

Figure 8: Load Balancer Pools

To ensure service availability in case of "Level 2" WAF failures, "Level 1"' servers voluntarily terminate their HTTP server if they find their counterpart to be unavailable. Monitoring is performed by sending HTTP requests every 30s to the level 2 WAFs with IP 10.0.1.49 and 10.0.1.28 using the monit tool [32].

Virtual servers were configured and assigned the pools previous created for HTTP and HTTPS service. The public IP address for each virtual server is 46.20.43.123 which is a redmine application DNS record.

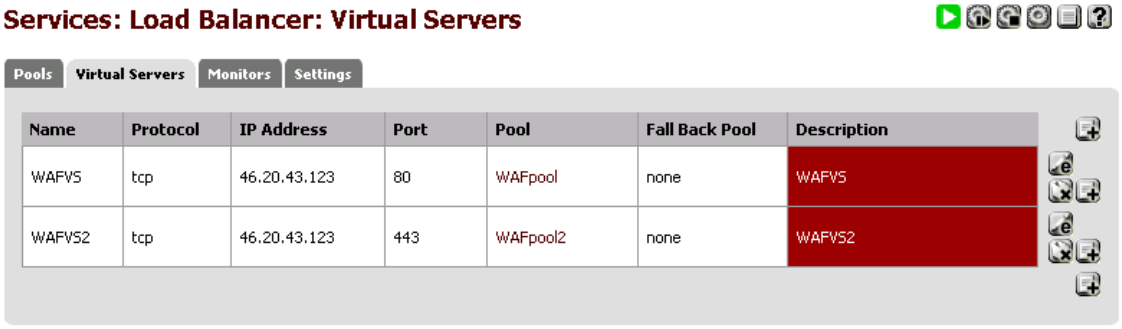

Figure 9: Load Balancer Virtual Servers

Chapter 4

Tests and Performance Evaluation

The performance of the WAF solution was evaluated and compared using a testbed that permits to assess naxsi and modsecurity separately and compose them in the target architecture. The evaluation focused on the measure the security effectiveness, throughput, and stability, reliability and usability. The security effectiveness represents the capability to detect specific attacks. Throughput measured the transactions speed in the current and proposed architecture to compare the overhead caused by naxsi and modsecurity. The stability and reliability measure the WAF's capability to distinguish legitimate from malicious traffic.

4.1 Security Effectiveness

To validate and verify the effectiveness of security controls provided with the redundant WAF architecture composed by naxsi and modsecurity we used security testing tools for web applications. In particular the OWASP Zed Attack Proxy (ZAP), Open Vulnerability Assessment System (OpenVAS) and burp suite.

We also evaluated security by executing customized security tests taking in consideration the vulnerabilities reported in Redmine Security Advisories [26] until 12/03/2016.

4.1.1 Security Testing Tools

The OWASP ZAP is a popular free security tool. It can help to automatically find security vulnerabilities in web applications or can be used for manual security testing. OWASP is a worldwide not-for-profit charitable organization focused on improving the security of software that counts with a number of relevant supporters such as Google and Microsoft. The OWASP 10 security vulnerabilities list [15] is a reference for many

security testing tools like Qualys Guard[17], Acunetix[1] and Nexpose[21]. However, all of them do not have a free edition. For this reason, we did not use in this project.

In addition to OWASP, a free edition of burp suite provided by Portwigger Ltd was used. This platform supports the entire security testing process, from initial mapping and analysis of an application's attack surface to finding and exploiting security vulnerabilities.

Kindly, OpenVAS is a free framework of several services and tools offering a vulnerability scanning and vulnerability management solution. To scan web applications, it integrates with nikto [26] and w3af [26] tools. OpenVAS creates a comprehensive way to explore and combine these free tools.

4.1.2 Test without WAF protection

Using OWASP ZAP we run a test against the redmine application. To configure a redmine user authentication we used the OWASP ZAP proxy feature. The results obtained are presented below.

Risk Level	Number of Alerts
High	1
Medium	0
Low	3

Table 4: Test without WAF protection

Was found one high-risk level vulnerabilities, which is XSS (persistent). The customized tests section explains how to exploit this vulnerability in the absence of WAF protection.

The low-level vulnerabilities are incomplete or no cache-control and pragma HTTP header set (settings), the absence of anti-CSRF Tokens and the allowance of password autocomplete in the browser.

The incomplete or no cache-control and pragma HTTP header set checks if cache-control HTTP header is set with no-cache, no-store, must-revalidate, private and that the pragma HTTP header is set with no-cache. If any of them is missing, it is considered incomplete and thus triggers the alert. In this case, the response from the server has two cache-control headers as we can see in the following response from the HTTP server:

```
HTTP/1.1 200 OK
Date: Fri, 05 Feb 2016 12:01:08 GMT
Server: Apache/2.2.8 (Ubuntu) mod_ssl/2.2.8 OpenSSL/0.9.8g
Phusion_Passenger/2.2.5
X-Powered-By: Phusion Passenger (mod_rails/mod_rack) 2.2.5
X-Runtime: 0.01472
ETag: "e09e73360faa05bcde95af220303dd9d"
Cache-Control: private, max-age=0, must-revalidate
Set-Cookie: _session_id=d06d292cc85d910ebb59f7faf956d06a; path=/;
HttpOnly; secure
X-Frame-Options: SAMEORIGIN
Content-Length: 3535
Status: 200 OK
X-XSS-Protection: 1; mode=block
Cache-Control: max-age=31536000, private, no-cache, no-store, must-
revalidate
Pragma: no-cache
Expires: 0
X-Content-Type-Options: nosniff
```

Figure 10: Cache-control Headers

This problem was reported to OWASP ZAP support team because the server was configured with the correct parameters on the apache main configuration file. Our report raise issue on GitHub [28] so the cache control scanner can accommodate multiple headers.

The absence of anti-CSRF Tokens means that the scanner did not find known anti-CSRF token. This redmine deployment does not implement CSRF token and the scanner detected it.

The password autocomplete in browser vulnerability means that the autocomplete attribute is not disabled in the redmine form "_form.rhtml". Consequently, the form Passwords may be stored in browsers and retrieved.

The OpenVAS framework test did not find any web vulnerability. It founds some low operating system vulnerabilities, which is out of the scope of this project. Compared with OpenVAS, the OWASP ZAP proxy features improves the detection of vulnerabilities such as XSS and CSRF because of the ability to collect data inserted in redmine pages.

4.1.3 Test with naxsi protection

Using OWASP ZAP we executed an authenticated test against the redmine application, obtaining the results presented below.

Risk Level	Number of Alerts
High	0
Medium	0
Low	3

Table 5: Test with Naxsi protection

This test identified three vulnerabilities equally reported without naxsi: password autocompletes in the browser, the absence of anti-CSRF tokens and incomplete or no cache-control and pragma HTTP header. The results show significant improvements in the web application security by implementing a naxsi WAF given that the high-risk alerts have been eliminated. The naxsi rules do not include the detection and prevention of CSRF and consequently, cannot protect the web application against this type of attack. The incomplete or no cache-control and pragma HTTP header refers to redmine default error pages returned as a response to unexpected requests. An important aspect of secure application development is to prevent error messages that give an attacker great insight into the inner workings of an application. It is important to assure the application fails safely under all possible error conditions, expected and unexpected.

The OpenVAS scan did not report any web vulnerability. It founds some low operating system vulnerabilities, which are out of the scope of this project.

4.1.4 Test with modsecurity protection

To be able to scan the application OWASP ZAP requires disabling modsecurity configuration rules with id 960015 and 960021. The first rule checks to see if an accepted header is present. The second rule checks if an accepted header exists, but is empty. The vulnerabilities found by OWASP when used modsecurity were:

Risk Level	Number of Alerts
High	0
Medium	0
Low	2

Table 6:Test with Modsecurity Protection

The scan results show two low-level vulnerabilities namely: password autocompletes in the browser and the absence of anti-CSRF tokens.

The password autocomplete in browser vulnerability was equally detected with naxsi WAF and the solution is the same as explained.

Modsecurity has a rule for CSRF which is part of the optional rules set. We enabled this rule to protect a critical location (/admin). When entering at this location, a CSRF Token is assigned to the request. However, modsecurity generates the warning below:

```
[Wed May 11 23:51:11.367462 2016] [:error] [pid 987:tid
140478968223488] [client 46.20.43.122] ModSecurity: Warning. Match
of "streq %{SESSION.CSRF_TOKEN}" against "ARGS:CSRF_TOKEN" required.
[file
"/usr/local/apache/conf/crs/activated_rules/modsecurity_crs_43_csrf_
protection.conf"] [line "34"] [id "981144"] [msg "CSRF Attack
Detected - Invalid Token."] [hostname "moldtrack.ai-emea.com"] [uri
"/admin"] [unique_id "VzOpTwoAARwAAAPbiP4AAAAD"]
```

Figure 11: Modsecurity CSRF ATTACK Log

The ideal is that the CSRF token should come from the application and not the WAF. However, modsecurity does have a method of inserting them, which is an interesting proof of concept. The modsecurity rule 981145 uses content injection to inject a bit of javascript to set the CSRF token on any URLs or forms for any HTML pages requested and when the subsequent request comes in after rule 981144 compares the CSRF Token and rejects any requests without a valid token.

As before, the OpenVAS scan did not report any web vulnerability. It founds some low operating system vulnerabilities, which are out of the scope of this project.

4.1.5 Test with Naxsi and Modsecurity Composition

The sequence composition considered both with naxsi in front of modsecurity and vice-versa. We verified that the results were the same in both implementations and are presented below.

Using OWASP ZAP we executed an authenticated test against the redmine application. The results are presented below.

Risk Level	Number of Alerts
High	0
Medium	0
Low	2

Table 7:Test with Naxsi and Modsecurity Composition

The results show two low-level vulnerabilities: password autocomplete in the browser and the absence of anti-CSRF tokens. This was predictable because the vulnerability scanner previously found this vulnerability in application code when testing the protection provided by naxsi and modsecurity separately.

Until now, these results suggest that there is no considerable security improvement when WAFs are combined. It should be noted that the Level 1 WAF could hide the presence of Level 2 WAF. This creates additional challenges to the adversary that may show beneficial in customized attacks scenarios.

The OpenVAS scan did not report any web vulnerability. It founds some low operating system vulnerabilities, which are out of the scope of this project.

The next step is to evaluate the solution proposed by executing customized attacks.

4.1.6 Customized tests

The customized tests were designed taking into consideration the list of security vulnerabilities from redmine security advisories [27] as of 15/03/2016.

The redmine security advisories contained some vulnerabilities not included in these tests:

- Data disclosure in atom feed
- Potential changeset message disclosure in issues API
- Data disclosure on the time logging form
- Potential data leak (project names) in the invalid form authenticity token error screen
- Ruby on Rails vulnerability (announcement)
- Mass-assignment vulnerability that would allow an attacker to bypass part of the security checks.

For these vulnerabilities, the most appropriate solution would be to apply the patches that have been made available, upgrade packages or recommended configurations. Some of the vulnerabilities related do API does not affect the redmine application because are not in use.

The table below shows the test results for each vulnerability considered for customized tests:

Severity	Details	Without WAF	Naxsi	Modsecurity	Naxsi+Modsecurity
Moderate	Open Redirect vulnerability	If we execute the following URL: https://moldtrack.ai-emea.com/login?back_url=@fakemoldtrack.com the user is redirected to fakemoldtrack.com.	This attack is blocked.	This attack is not blocked.	This attack is blocked.
High	Persistent XSS vulnerability	In project settings ->version. The version field does not make input validation. If we insert a script like <script>alert("XSS")</script> in the name field, when the user clicks on roadmap the browser will execute the script inserted on the name.	This attack is blocked.	This attack is blocked.	This attack is blocked.
High	Vulnerability that would allow an attacker to bypass the CSRF protection	Using the HTML code (appendix B) it is possible to create a user with admin privileges. The session must be from a user with permissions to create users.	This attack is not blocked.	This attack is not blocked.	This attack is not blocked.

From OWASP ZAP and customized tests, we see that combination of both WAF's improves the protection of redmine application over the possible attacks. The table below depicts a comparison between the number of vulnerabilities when not using a WAF protection, using naxsi or modsecurity separately and combining both WAFs.

Test	No WAF	Naxsi	modsecurity	Naxsi+Modsecurity
OWASP ZAP	4	3	2	2
Customized	3	1	2	1
Distinct	5	3	3	2

Table 8: Vulnerabilities Identified in OWASP ZAP and Customized Tests

The last row counts the number of distinct vulnerabilities after remaining those that were found in both experiments.

The solution proposed has less number of vulnerabilities because it takes the benefits from naxsi and modsecurity. It is expected that naxsi and modsecurity capabilities will be improved to accommodate new threats and needs. This updates may happen in different periods and approaches considering that distinct developers perform it. Consequently, our solution will take advantage of the newly developed features from both naxsi and modsecurity.

4.1 Performance

The next step for the comparison analysis evaluates the throughput for each of the test cases. To collect data for the performance analysis we used the visual studio web performance and load tests tool. It is a product from Microsoft, which helps to determine how well a software responds to various levels of usage. It allows a simulation of multiple users opening simultaneous connections to a server and making multiple HTTP requests.

The objective is to determine how much overhead the WAF places in the system.

We measured the times for the following applications pages: open the webpage, login, list projects, browse a specific project, create a new issue in a specific project with a document upload, download a file from the specific project, show my page and use the search option.

The page loading times were measured in six different configurations with caching disabled in the proxies:

- Without a WAF
- Naxsi
- Modsecurity
- Naxsi and modsecurity in sequence
- Testbed
- Testbed under attack. To evaluate this scenario, we execute the OWSAP ZAP simultaneous with the visual studio web performance and load tests tool

Each of the tests duration was 600s and simulated 25 concurrent users.

The first step of the performance test was the definition of the requests to be performed by using the visual studio record option. This is a plugin, which records the browser requests and captures their parameters. To make sure that the requests executed successfully by the application we monitor the redmine' production log file:

```
$ tail -f production.log

Processing AttachmentsController#show (for 46.20.43.122 at 2016-03-07
17:10:14) [GET]
  Session ID: cf4eb64fa36df6fce81928082b13157c
  Parameters:             {"action"=>"show",             "id"=>"432",
"controller"=>"attachments", "filename"=>"Report_August.pdf"}
Completed in 0.04694 (21 reqs/sec) | Rendering: 0.00040 (0%) | DB:
0.04099     (87%)      |      200      OK      [https://moldtrack.ai-
emea.com/attachments/432/Report_August.pdf]
Streaming                                                        file
/var/www/railsapp/files/140925095819_Report_August.pdf
```

Figure 12: Redmine Log

From the production log excerpt above, we can see an example of a request to download a file "Record_August.pdf" which was executed successfully (response code "200").

The tests for each scenario returned the following data:

- Page: the name of the page used in the test
- Total: the total number of requests made for each page.
- Avg: the average page response time
- Min.: the minimum page response time
- Median: the median page response time

- 99 %: the 99th percentile for the response time. This indicates that 99% of the pages responded faster than this value
- Max.: the maximum page response time
- Std. Dev.: the standard deviation of the pages response times

The table below depicts the metrics above when no WAF was used. All values are in seconds.

Page	Total	Avg.	Min.	Median	99%	Max.	Std. Dev.
https://moldtrack.ai-emea.com/	100	1.930	0.031	0.200	9.450	9.450	3.110
/attachments/432/Report_August.pdf	75	0.410	0.034	0.055	4.000	4.000	0.930
/issues/show/1219	174	3.970	0.700	3.610	12.500	12.800	2.730
/login {POST}	99	1.260	-	-	17.500	17.500	3.210
/my/page	75	0.720	0.081	0.260	4.950	4.950	1.040
/projects	75	0.260	0.046	0.089	4.370	4.370	0.670
/projects/activity/antex-infra	79	0.560	0.120	0.500	1.690	1.690	0.320
/projects/antex-infra/documents	78	0.120	0.035	0.067	0.750	0.750	0.140
/projects/antex-infra/issues	154	0.760	0.160	0.450	6.040	6.570	1.050
/projects/antex-infra/issues/new {GET}	177	0.840	0.060	0.170	8.420	11.000	1.890
/projects/antex-infra/issues/new {POST}	80	0.940	0.230	0.580	4.310	4.310	0.860
/projects/settings/antex-infra	78	0.210	0.067	0.140	1.100	1.100	0.210
/projects/show/antex-infra	154	0.250	0.046	0.120	3.320	5.830	0.600
/search/index/antex-infra	75	0.390	0.057	0.280	4.500	4.500	0.560

Table 9: Response Time without WAF

The Min. and Median metrics were not possible to calculate accurately for the login page in all scenarios, even after repeatedly execute the measurement. This issue was found for all performance tests as we can see in remain results.

The next table depicts the metrics obtained when a naxsi WAF was deployed between the HTTPS client and the application.

Page	Total	Avg.	Min.	Median	99%	Max.	Std. Dev.
https://moldtrack.ai-emea.com/	99	2.470	0.037	0.170	10.600	10.600	3.810
/attachments/432/Report_August.pdf	74	0.420	0.045	0.072	4.750	4.750	0.960
/issues/show/1219	167	4.880	0.810	3.910	19.900	22.500	4.070
/login {POST}	96	1.180	-	-	8.240	8.240	2.420
/my/page	75	0.440	0.120	0.300	2.750	2.750	0.490
/projects	75	0.410	0.095	0.190	4.850	4.850	0.730
/projects/activity/antex-infra	78	1.880	0.680	1.660	4.090	4.090	0.870
/projects/antex-infra/documents	78	0.240	0.094	0.190	1.580	1.580	0.220
/projects/antex-infra/issues	152	1.010	0.230	0.760	5.800	6.770	0.980
/projects/antex-infra/issues/new {GET}	169	0.980	0.150	0.330	15.700	16.400	2.560
/projects/antex-infra/issues/new {POST}	82	1.050	0.330	0.770	5.630	5.630	0.810
/projects/settings/antex-infra	78	0.490	0.120	0.350	1.830	1.83	0.380
/projects/show/antex-infra	153	0.450	0.091	0.220	5.610	7.49	0.940
/search/index/antex-infra	74	0.570	0.120	0.370	4.650	4.65	0.620

Table 10: Response Time with Naxsi

The table below shows the metrics for the modsecurity configuration.

Page	Total	Avg.	Min.	Median	99%	Max.	Std. Dev.
https://moldtrack.ai-emea.com/	99	2.230	0.043	0.150	9.180	9.180	3.550
/attachments/432/Report_August.pdf	74	0.590	0.047	0.100	5.580	5.580	1.240
/issues/show/1219	168	4.480	0.810	4.070	12.900	18.300	3.070
/login {POST}	95	1.780	-	-	19.500	19.500	4.200
/my/page	75	0.470	0.120	0.310	4.440	4.440	0.630
/projects	75	0.280	0.093	0.170	2.280	2.280	0.310
/projects/activity/antex-infra	77	3.200	0.810	3.110	7.420	7.420	1.670
/projects/antex-infra/documents	77	0.360	0.097	0.200	2.150	2.150	0.390
/projects/antex-infra/issues	152	1.070	0.230	0.830	6.030	6.920	0.920
/projects/antex-infra/issues/new {GET}	169	1.000	0.150	0.400	8.550	10.900	1.700
/projects/antex-infra/issues/new {POST}	80	1.230	0.330	0.960	4.990	4.990	0.870
/projects/settings/antex-infra	77	0.570	0.120	0.380	4.710	4.710	0.650
/projects/show/antex-infra	152	0.480	0.093	0.220	5.310	5.780	0.860
/search/index/antex-infra	74	0.420	0.110	0.320	2.580	2.580	0.420

Table 11: Response Time with Modsecurity

The next table presents the metrics when both naxsi and modsecurity are in place.

Page	Total	Avg.	Min.	Median	99%	Max.	Std. Dev.
https://moldtrack.ai-emea.com/	90	3.300	0.042	0.190	17.200	17.200	5.360
/attachments/432/Report_August.pdf	65	1.000	0.055	0.110	5.620	5.620	1.660
/issues/show/1219	153	5.250	0.930	4.330	15.000	18.300	3.590
/login {POST}	88	2.210	-	-	22.200	22.200	5.030
/my/page	73	0.500	0.150	0.260	4.170	4.170	0.630
/projects	73	0.510	0.120	0.230	7.660	7.660	1.020
/projects/activity/antex-infra	76	3.260	0.700	3.170	8.680	8.680	1.820
/projects/antex-infra/documents	76	0.610	0.120	0.280	4.400	4.400	0.840
/projects/antex-infra/issues	147	1.520	0.260	1.060	6.650	7.570	1.410
/projects/antex-infra/issues/new {GET}	161	1.350	0.180	0.520	8.380	11.400	1.890
/projects/antex-infra/issues/new {POST}	78	1.440	0.410	1.170	7.010	7.010	1.080
/projects/settings/antex-infra	76	0.950	0.140	0.600	5.480	5.480	1.090
/projects/show/antex-infra	149	0.600	0.110	0.300	4.490	6.310	0.910
/search/index/antex-infra	65	0.720	0.140	0.500	4.920	4.920	0.760

Table 12: Response Time with Naxsi and Modsecurity

The next table depicts the results obtained the architecture with one load balancer, two naxsi nodes, and two modsecurity nodes.

Page	Total	Avg.	Min.	Median	99%	Max.	Std. Dev.
https://moldtrack.ai-emea.com/	100	1.540	0.041	0.160	4.410	12.500	2.750
/attachments/432/Report_August.pdf	75	0.680	0.046	0.140	2.040	5.550	1.100
/issues/show/1219	173	4.420	0.460	3.590	8.560	15.700	3.180
/login {POST}	98	1.350	-	-	6.050	22.500	3.650
/my/page	75	0.670	0.130	0.290	1.010	7.230	1.290
/projects	75	0.550	0.089	0.220	0.900	7.180	1.110
/projects/activity/antex-infra	81	0.470	0.150	0.350	0.810	1.770	0.330
/projects/antex-infra/documents	80	0.270	0.097	0.220	0.460	1.650	0.210
/projects/antex-infra/issues	155	1.030	0.230	0.730	1.860	7.750	1.190
/projects/antex-infra/issues/new {GET}	178	0.920	0.140	0.370	1.610	10.900	1.720
/projects/antex-infra/issues/new {POST}	81	1.600	0.460	1.140	3.720	5.180	1.210
/projects/settings/antex-infra	80	0.410	0.120	0.320	0.840	2.110	0.330
/projects/show/antex-infra	156	0.390	0.095	0.210	0.640	3.940	0.590
/search/index/antex-infra	75	0.630	0.120	0.450	1.320	4.580	0.650

Table 13: Response Times with the Architecture proposed

Kindly, the results for the scenario where the system is under attack are depicted below.

Page	Total	Avg.	Min.	Median	99%	Max.	Std. Dev.
https://moldtrack.ai-emea.com/	94	2.010	0.043	0.310	7.590	7.590	2.530
/attachments/432/Report_August.pdf	69	1.170	0.057	0.340	11.500	11.500	2.040
/issues/show/1219	151	6.600	0.920	5.970	16.800	17.200	3.450
/login {POST}	86	2.390	-	-	27.400	27.400	5.260
/my/page	75	1.070	0.150	0.510	5.430	5.430	1.300
/projects	75	0.950	0.096	0.350	6.210	6.210	1.420
/projects/activity/antex-infra	75	1.440	0.330	1.420	3.220	3.220	0.820
/projects/antex-infra/documents	75	0.600	0.110	0.320	5.980	5.980	0.830
/projects/antex-infra/issues	147	1.740	0.270	1.280	7.870	10.200	1.630
/projects/antex-infra/issues/new {GET}	157	1.800	0.160	0.810	10.800	16.800	2.510
/projects/antex-infra/issues/new {POST}	77	2.860	0.500	2.040	11.800	11.800	2.490
/projects/settings/antex-infra	75	0.870	0.150	0.480	7.690	7.690	1.090
/projects/show/antex-infra	148	0.780	0.120	0.400	6.880	7.420	1.160
/search/index/antex-infra	69	1.120	0.170	0.560	9.930	9.930	1.550

Table 14: Response Times with the Architecture proposed under Attack

For comparison purpose, the table below aggregates the average response time for No WAF, naxsi (N), modsecurity (M), naxsi and modsecurity in sequence (N+M), testbed and testbed under attack.

	Average page response time					
Page	**No WAF**	**N**	**M**	**N + M**	**Testbed**	**Testbed under Attack**
https://moldtrack.ai-emea.com/	1.930	2.470	2.230	3.300	1.540	2.010
/attachments/432/Report_August.pdf	0.410	0.420	0.590	1.000	0.680	1.170
/issues/show/1219	3.970	4.880	4.480	5.250	4.420	6.600
/login {POST}	1.260	1.180	1.780	2.210	1.350	2.390
/my/page	0.720	0.440	0.470	0.500	0.670	1.070
/projects	0.260	0.410	0.280	0.510	0.550	0.950
/projects/activity/antex-infra	0.560	1.880	3.200	3.260	0.470	1.440
/projects/antex-infra/documents	0.120	0.240	0.360	0.610	0.270	0.600
/projects/antex-infra/issues	0.760	1.010	1.070	1.520	1.030	1.740
/projects/antex-infra/issues/new {GET}	0.840	0.980	1.000	1.350	0.920	1.800
/projects/antex-infra/issues/new {POST}	0.940	1.050	1.230	1.440	1.600	2.860
/projects/settings/antex-infra	0.210	0.490	0.570	0.950	0.410	0.870
/projects/show/antex-infra	0.250	0.450	0.480	0.600	0.390	0.780
/search/index/antex-infra	0.390	0.570	0.420	0.720	0.630	1.120

Table 15: Average Response Time Comparison

The table confirms that any architecture using WAF introduce some delay in the access to redmine pages. It should be noted that these results were conducted with caching disabled. We follow the recommendations from VirtualBox bug report regarding caching files by nginx and apache [36]. Apache and Nginx use mmap() for fast access to static content. Probably this mechanism needs a special support of the underlying file system which vboxvfs does not provide.

Running the tests with cache enabled would contribute to an improved performance because some operations are delivered to the WAFs. However, this would introduce a bias in the experiments as they were built to precisely evaluate WAF's performance.

The results evidence two anomalies with the average response time of a login page (/login {POST}) and of a web page running a database query (/my/page) being consistently lower when naxsi and modsecurity are used. The results were observed repeatedly in the multiple experiments performed to identify the reason, although the root cause was not found.

A side-by-side comparison of Naxsi and Modsecurity shows no clear winner, with each of the WAFs presenting an improved performance on some of the web pages tested. Relevant for this study is the comparison of the composition of both WAFs with and without a load balancer, which evidence that the introduction of the later can absorb the negative performance penalty of the former by equally distributing the load by the two WAF circuits defined.

Below is represented the average response time in graphical mode. It shows that some operations such as “show issues for specific project” take considerable time in all scenarios.

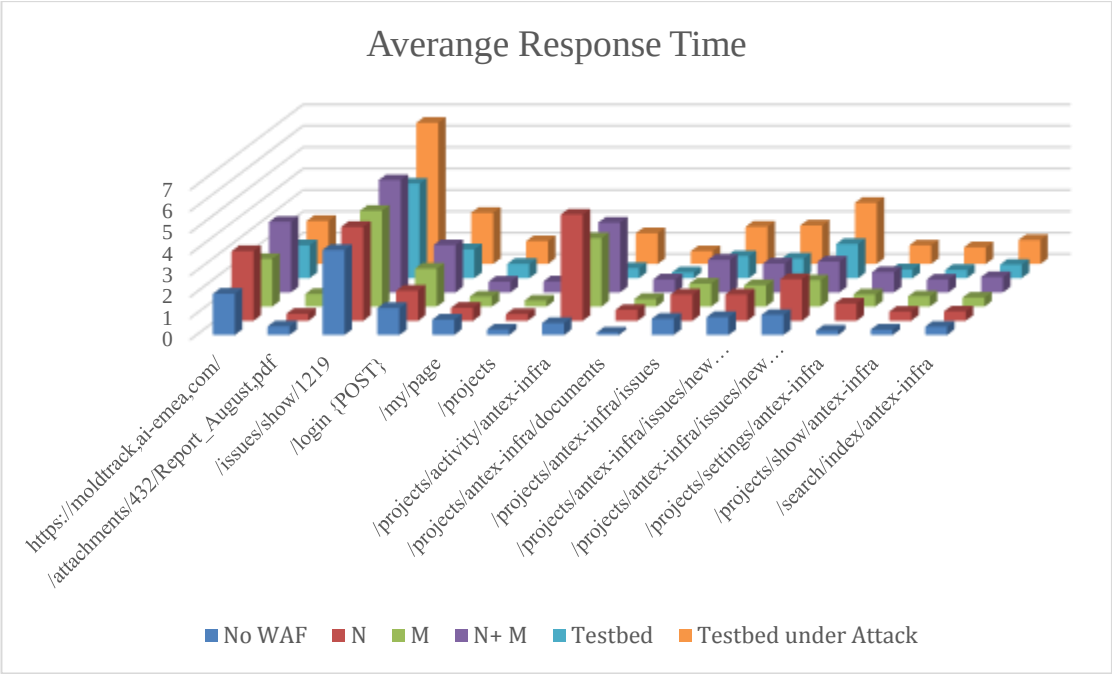

Figure 13: Average Response Time Graphic

The standard deviation of the response time metric, depicted in Figure 14 clearly shows that the request type is the most influencing factor in the predictability of the response time. The exception is the Naxsi+Modsecurity composition when the load balancer is not used which consistently presents an above average standard deviation.

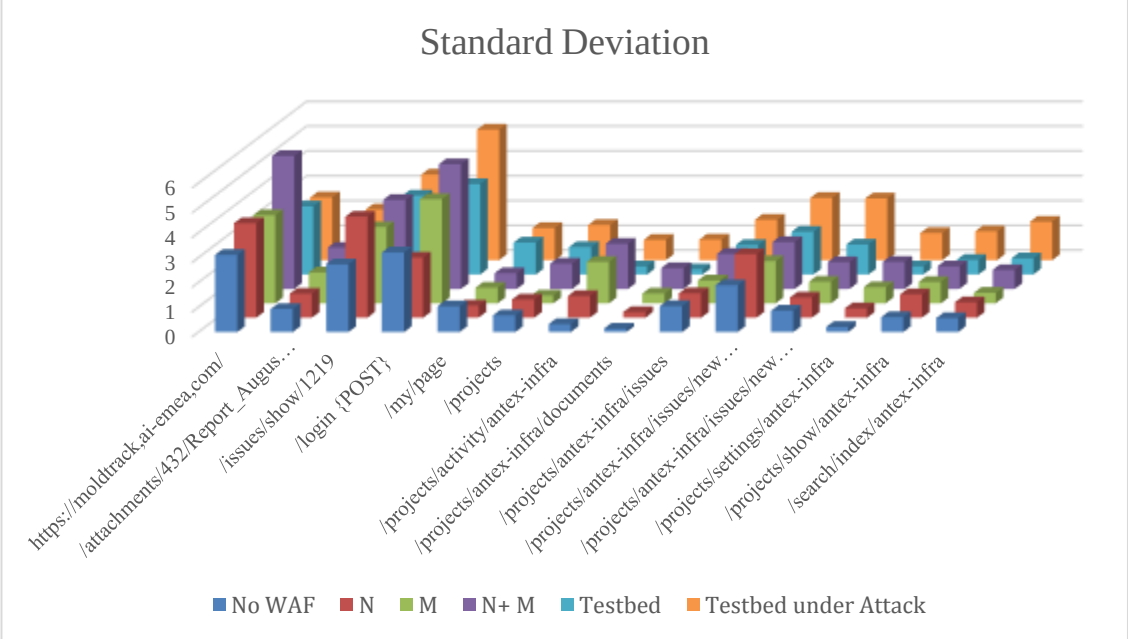

Figure 14: Standard Deviation Graphic

4.1 Stability and reliability and usability

A comparison of the functional and usability characteristics of both naxsi and modsecurity shows that naxsi does not rely upon pre-defined signatures, so it should be capable of defeating complex, unknown, obfuscated attack patterns. Naxsi by default reads a small subset of simple (and readable) rules containing known patterns involved in website vulnerabilities. For example, <, | or drop expression are not supposed to be part of a URI. This approach is very efficient, but very prone to false positives as well. However, it comes with a helpful tool (nxtool) to assist the "learning" process and which facilitates the identification of false positives and the definition of rules avoid them. This phase is important to avoid the risk of locking out legitimate users from the application. In terms of usability, the rules are very simple and easy to understand.

Modsecurity also generates many false-positive results, as these rules are standard and not application-friendly. It is necessary to edit the default rules in order to eliminate the false positive results. Alternatively, it is possible to create a whitelist rules according to the application's requirement, something that it was not required in the scope of this work. The rule creation is complex but can be extended or changed as needed. Modsecurity uses the popular Perl Compatible Regular Expressions (PCRE) library for pattern matching which with the advantage of being available for several operating systems.

The creation of whitelists should be handled with care to avoid the opening of windows of exploitation for an adversary. Unfortunately, this requires particular caution and a permanent monitoring of the logs.

Chapter 5

Conclusion and Future Work

The traditional firewall architectures improve availability by using redundant nodes with the exact same characteristics and configurations. One of their main benefits is to facilitate the systems administration activities. This strategy is equally beneficial from an economic perspective as it reduces the training investment, which becomes constrained to one technology.

However, redundancy does not contribute to improving security. Instead, systems should be heterogeneous as diversification enables the servers to be more resilient to attacks because the probability of using the same exploit to compromise diverse systems is lower.

This work investigated the contribution of a combination of two popular open source WAF, naxsi and modsecurity to protect a web infrastructure. Both WAFs have the ability to understand the application level content.

The naxsi and modsecurity differ in the following: learning mode, buffering and virtual patching. Using both WAFs in the architecture, we can take advantage of all the features available.

The security test results show that the combination of naxsi and modsecurity offer a better protection against OWASP ZAP and customized tests. This fact is a great advantage against high skilled adversaries. They combine multiple attack methodologies and tools in order to reach and compromise their target.

Even thought our solution ensures a more secure environment, the WAF´s affects the web service performance. This fact is a drawback of increasing the level of security. As expected, our proposed architecture adds extra overhead to the response time of the requests due to extra addition request processing. However, the results showed that the user experience could still be considered acceptable.

Further work may focus on improving the security of the proposed solution by implementing the rejuvenation principle and a combination of dynamic application security testing (DAST). The rejuvenation implies that the nodes are periodically restored to the last corrected known image. The architecture and technology used in this

architecture facilitate the implementation of this concept. A virtual machine can be easily restored from a snapshot and the redundant node ensure the service continuity while the other node is rejuvenated.

The DAST is a process of security testing an application or software product in a running state. A DAST scanner (Burp, OWASP Zed Attack Proxy) generates a report that serves as an input for WAF signatures.

Bibliography

1. Acunetix. (n.d.). Web application security with Acunetix. Retrieved October 28, 2015, from https://www.acunetix.com.
2. Barnett, R. (2009). Waf virtual patching challenge: Securing webgoat with modsecurity. *Breach Security*.
3. Bessani, A., Daidone, A., Gashi, I., Obelheiro, R., Sousa, P., & Stankovic, V. (2009, June). Enhancing fault/intrusion tolerance through design and configuration diversity. In *Proceedings of the 3rd Workshop on Recent Advances on Intrusion-Tolerant Systems–WRAITS* (Vol. 9).
4. Carvalho R. & Correia M. (2014, September). Authentication Security through Diversity and Redundancy for Cloud Computing. In *INForum 2014 - Atas do 6º Simpósio de Informática.*
5. Dermann, M., Dziadzka, M., Hemkemeier, B., Hoffmann, A., Meisel, A., Rohr, M., & Schreiber, T. (2008). Best practices: Use of web application firewalls. *The Open Web Security Application Project*, OWASP Papers Program.
6. Desmet, L., Piessens, F., Joosen, W., & Verbaeten, P. (2006, November). Bridging the gap between web application firewalls and web applications. In *Proceedings of the fourth ACM workshop on Formal methods in security* (pp. 67-77). ACM.
7. Garcia, M., Bessani, A., Gashi, I., Neves, N., & Obelheiro, R. (2014). Analysis of operating system diversity for intrusion tolerance. *Software: Practice and Experience*, 44(6), 735-770.
8. Gehling, B., & Stankard, D. (2005, September). eCommerce security. In *Proceedings of the 2nd annual conference on Information security curriculum development* (pp. 32-37). ACM.
9. GitHub. (n.d.). Nbs-system/naxsi. Retrieved October 28, 2015, from https://github.com/nbs-system/naxsi
10. Kazanavicius, E., Kazanavicius, V., Venckauskas, A., & Paskevicius, R. (2012). Securing web application by embedded firewall. *Elektronika ir Elektrotechnika*, *119* (3), 65-68.

11. Larsen, P., Brunthaler, S., & Franz, M. (2014). Security through diversity: Are we there yet?. *Security & Privacy, IEEE*, *12* (2), 28-35.
12. Littlewood, B., & Strigini, L. (2004). Redundancy and diversity in security. In *Computer Security –ESORICS 2004* (pp. 423-438). Springer Berlin Heidelberg.
13. NSS Labs. (2013). TEST METHODOLOGY Web Application Firewall. Retrieved November 9, 2015, from https://library.nsslabs.com/sites/default/files/public-report/files/Web Application Firewall Test Methodology v6_2.pdf.
14. OWASP (2015, September 15). Web Application Firewall. Retrieved October 28, 2015, from https://www.owasp.org/index.php/Web_Application_Firewall.
15. OWASP. (2015, October 7). OWASP Zed Attack Proxy Project. Retrieved October 28, 2015, from https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project.
16. OWASP https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
17. Qualys. (n.d.). The Next Generation Cloud Security Platform. Retrieved October 28, 2015, from https://www.qualys.com.
18. Saidane, A., Nicomette, V., & Deswarte, Y. (2009). The design of a generic intrusion-tolerant architecture for web servers. *Dependable and Secure Computing, IEEE Transactions on*, *6*(1), 45-58.
19. Raj, S., & Varghese, G. (2011, March). Analysis of intrusion-tolerant architectures for Web Servers. In *Emerging Trends in Electrical and Computer Technology (ICETECT), 2011 International Conference on* (pp. 998-1003). IEEE.
20. Razzaq, A., Hur, A., Shahbaz, S., Masood, M., & Ahmad, H. F. (2013, March). Critical analysis on web application firewall solutions. In *Autonomous Decentralized Systems (ISADS), 2013 IEEE Eleventh International Symposium on* (pp. 1-6). IEEE.
21. Rapid7. (n.d.). Penetration Testing Software. Retrieved October 28, 2015, from http://www.metasploit.com.

22. Trustwave. (n.d.). ModSecurity: Open Source Web Application Firewall. Retrieved October 28, 2015, from https://www.modsecurity.org.
23. W3Techs. (n.d.). Current CVSS Score Distribution For All Vulnerabilities. Retrieved November 17, 2015, from https://www.cvedetails.com.
24. Symantec. (2014, April). INTERNET SECURITY THREAT REPORT. Retrieved November 17, 2015, from http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v19_21291018.en-us.pdf.
25. Bitbucket. (2016, March 29). Doxi-rules. Retrieved June 06, 2016, from https://bitbucket.org/lazy_dogtown/doxi-rules.
26. OpenVAS. (n.d.). Open Vulnerability Assessment System. Retrieved June 06, 2016, from http://www.openvas.org.
27. Redmine. (n.d.). Retrieved June 06, 2016, from https://www.redmine.org/projects/redmine/wiki/Security_Advisories.
28. Github. (2016, April 14). Incomplete or no cache-control and pragma HTTP header (False positive?) · Issue #2405 zaproxy/zaproxy. Retrieved June 06, 2016, from https://github.com/zaproxy/zaproxy/issues/2405https://github.com/zaproxy/zaproxy/issues/2405.
29. Lang, J. (2015, April 09). Redmine. Retrieved June 06, 2016, from http://www.redmine.org/issues/19577.
30. WAPT. (n.d.). Response Time. Retrieved June 06, 2016, from http://www.loadtestingtool.com/help/response-time.shtm.
31. Tymoshyk, N., & Breslavskyi, S. (2015, March 04). Web Application Firewalls: Next Big Thing in Security. Retrieved June 06, 2016, from http://www.esecurityplanet.com/network-security/web-application-firewalls-next-big-thing-in-security.html.
32. M/Monit. (n.d.). Easy, proactive monitoring of Unix. Retrieved June 06, 2016 from https://mmonit.com/.
33. Microsoft. (n.d.). Performance Testing Guidance for Web Applications. Retrieved June 06, 2016, from https://msdn.microsoft.com/en-us/library/bb924375.aspx.

34. Positive Technologies (2015). WEB APPLICATION VULNERABILITY STATISTICS 2014. Retrieved July 16, 2016, from https://www.ptsecurity.com/upload/ptcom/WEB_APP_VULNERABILITY_2014.A4.ENG.242465.14.OCT.2015.pdf.
35. https://www.virtualbox.org/ticket/819
36. Oracle. (2016, May 09). Ticket #819 (reopened defect). Retrieved July 19, 2016, from https://www.virtualbox.org/ticket/819.

Appendix A

Whitelist

```
BasicRule wl:1005 "mz:$HEADERS_VAR:cookie";
BasicRule wl:1010 "mz:$HEADERS_VAR:cookie";
BasicRule wl:1011 "mz:$HEADERS_VAR:cookie";
BasicRule wl:1315 "mz:$HEADERS_VAR:cookie";
BasicRule wl:1315 "mz:BODY";
BasicRule l:1315 "mz:$BODY_VAR:back_url";
BasicRule wl:1315 "mz:$URL_X:/login|$BODY_VAR:back_url";
BasicRule wl:2 "mz:$URL:/issues|BODY";
BasicRule wl:2 "mz:BODY";
BasicRule l:2 "mz:$URL_X:/issues|BODY";
BasicRule wl:1101 "mz:$URL_X:/login|$ARGS_VAR:back_url";
BasicRule wl:1101 "mz:$URL_X:/login|ARGS";
BasicRule wl:1316 "mz:$URL_X:/account|$ARGS_VAR:mail";
BasicRule wl:1316 "mz:$URL_X:/account|ARGS";
BasicRule wl:1007 "mz:$BODY_VAR:issue[due_date]";
BasicRule wl:1007 "mz:$BODY_VAR:issue[start_date]";
BasicRule wl:1007 "mz:$URL_X:/issues|$BODY_VAR:issue[start_date]";
BasicRule wl:1007 "mz:$URL_X:/issues|BODY";
BasicRulewl:1007 "mz:$URL_X:/issues|$BODY_VAR:issue[due_date]";
BasicRule wl:1007 "mz:$URL_X:/issues|$BODY_VAR:issue[due_date]";
BasicRule wl:1007 "mz:$URL_X:/issues|$BODY_VAR:issue[start_date]";
BasicRule l:1007 "mz:$URL_X:/issues|$BODY_VAR:time_entry[comments]";
BasicRule wl:1007 "mz:$BODY_VAR:time_entry[comments]";
BasicRule wl:1007 "mz:BODY";
BasicRule wl:1310 "mz:$URL_X:/issues|BODY|NAME";
BasicRule l:1311 "mz:$URL_X:/issues|BODY|NAME";
BasicRule wl:1311 "mz:$URL_X:/issues|$BODY_VAR:notes";
BasicRule wl:1311 "mz:BODY";
BasicRule wl:1311 "mz:$URL_X:/issues|BODY";
BasicRule wl:1311 "mz:$BODY_VAR:notes";
BasicRule wl:1000
"mz:$URL_X:/issues/edit|$BODY_VAR:issue[description]";
BasicRule wl:1000 "mz:BODY";
BasicRule wl:1000 "mz:$URL_X:/issues/edit|BODY";
BasicRule wl:1000 "mz:$BODY_VAR:issue[description]";
BasicRule wl:1015
"mz:$URL_X:/issues/edit|$BODY_VAR:issue[description]";
BasicRule wl:1015 "mz:BODY";
BasicRule wl:1015 "mz:$URL_X:/issues/edit|BODY";
BasicRule wl:1015 "mz:$BODY_VAR:issue[description]";
BasicRule wl:1310 "mz:$URL_X:/users/edit|BODY|NAME";
          BasicRule wl:1311 "mz:$URL_X:/users/edit|BODY|NAME";
```

Appendix B

CFRF Code

```
<html>
<body>
    <form method=POST action="https://moldtrack.ai-emea.com/users/add">
       <input  style="display:  none"  type="text"  value="adversary"
size="25" name="user[login]" id="user_login"/>
       <input  style="display:  none"  type="text"  value="adversary"
size="30" name="user[firstname]" id="user_firstname"/>
       <input  style="display:  none"  type="text"  value="adversary"
size="30" name="user[lastname]" id="user_lastname"/>
       <input  style="display:  none"  type="text"  value="adversary@ai-
emea.com" size="30" name="user[mail]" id="user_mail"/>
       <input   style="display:   none"   type="password"   size="25"
name="password" id="password" value="adversary" />
       <input   style="display:   none"   type="password"   size="25"
name="password_confirmation"                id="password_confirmation"
value="adversary" />
       <input   style="display:   none"   type="checkbox"   value="1"
name="user[admin]" id="user_admin"/>
       <input    style="display:    none"    type="hidden"    value="1"
name="user[admin]"/>
       <input  style="display:  none"  type="submit"  value="Create"
id="commit" name="commit" />
  </form>
  <script>document.getElementById("commit").click();</script>
</body>
</html>
```